

# Quantum Monte Carlo Simulations

Anouar Benali, Ph.D.

Assistant Computational Scientist

Argonne Leadership Computing Facility  
Argonne National Laboratory - Argonne, IL, USA

Contact: [benali@anl.gov](mailto:benali@anl.gov)



U.S. DEPARTMENT OF  
**ENERGY**

# Outline

- (Very) brief introduction to Electronic Structure
- Monte Carlo Simulation
- Quantum Monte Carlo (QMC) Simulations
  - VARIATIONAL MONTE CARLO (VMC)
  - DIFFUSION MONT CALRO (DMC)
  - TRIAL WAVEFUNCTION
- QMCPACK: Hybrid Parallelization Computational view
- Performance targeted for specific platform
- Examples of Applications



# Electronic Structure



# First-Principles Computational Physics/Chemistry

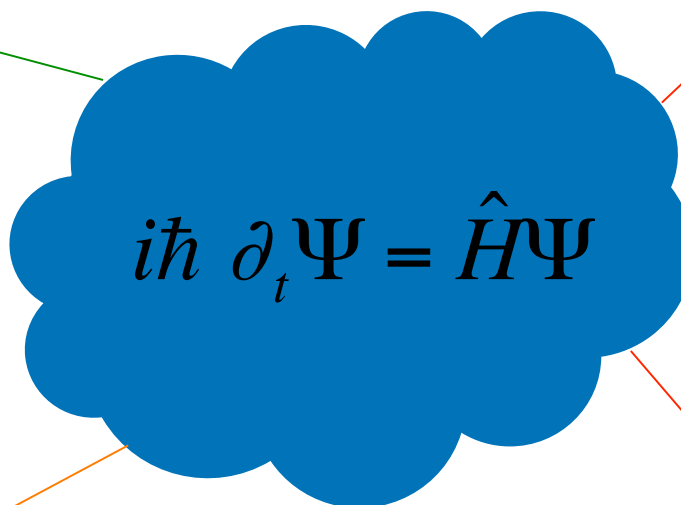
## Schrodinger Equation Theory of electrons and ions

### Chemistry:

- chemical reactions
- molecular properties
- bonding patterns/formation
- solvation properties
- surfaces

### Exotic electronic phases:

- Superconductivity
- Magnetism
- Charge ordering


$$i\hbar \partial_t \Psi = \hat{H} \Psi$$

### Thermodynamics:

- crystal structures
- phase transitions
- strength
- defects and dislocations

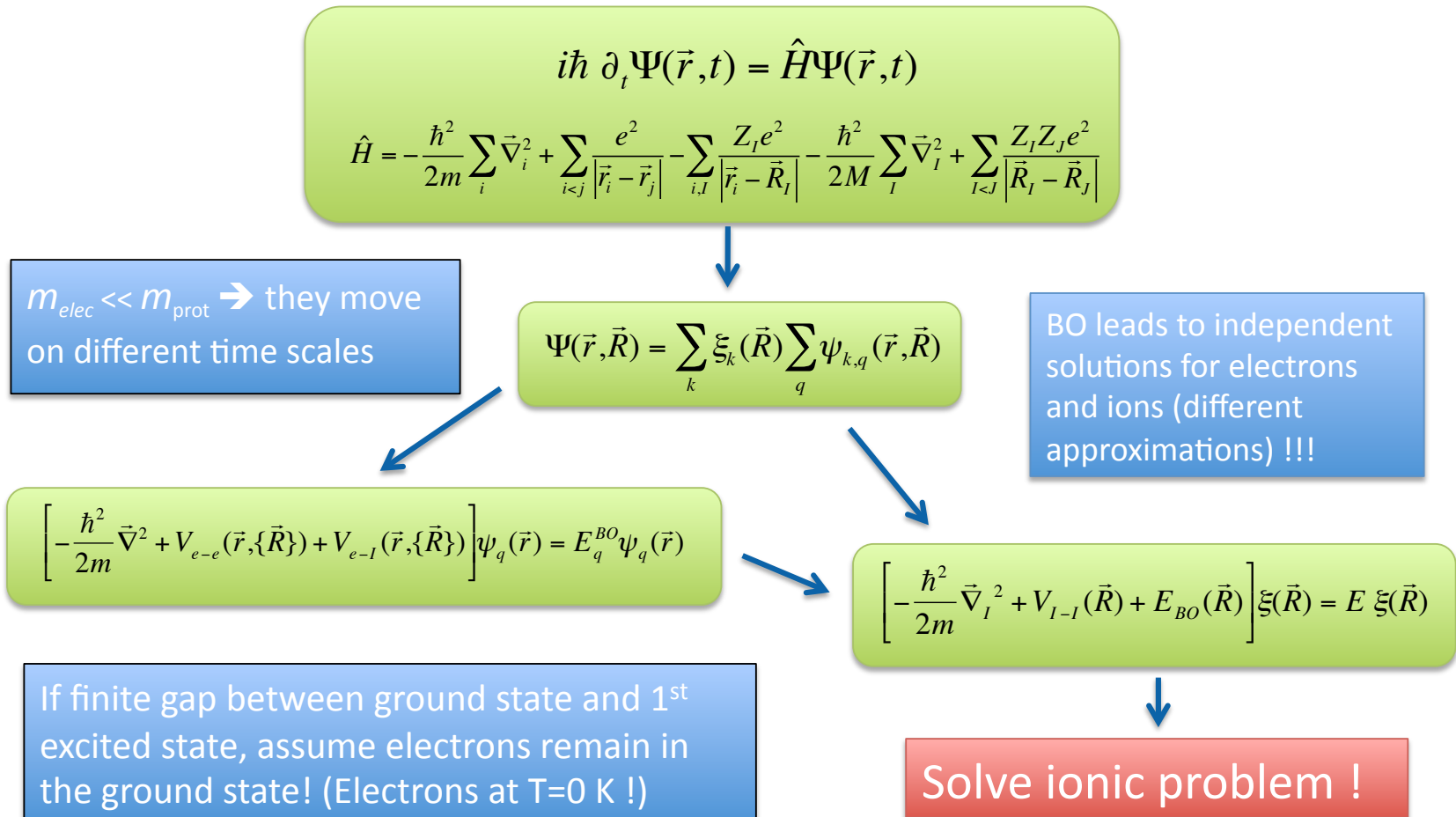
### Plasmas:

- equilibrium prop.
- kinetics
- instabilities
- shocks

### Optical and Transport Properties:

- spectra
- conductivity
- viscosity
- diffusion

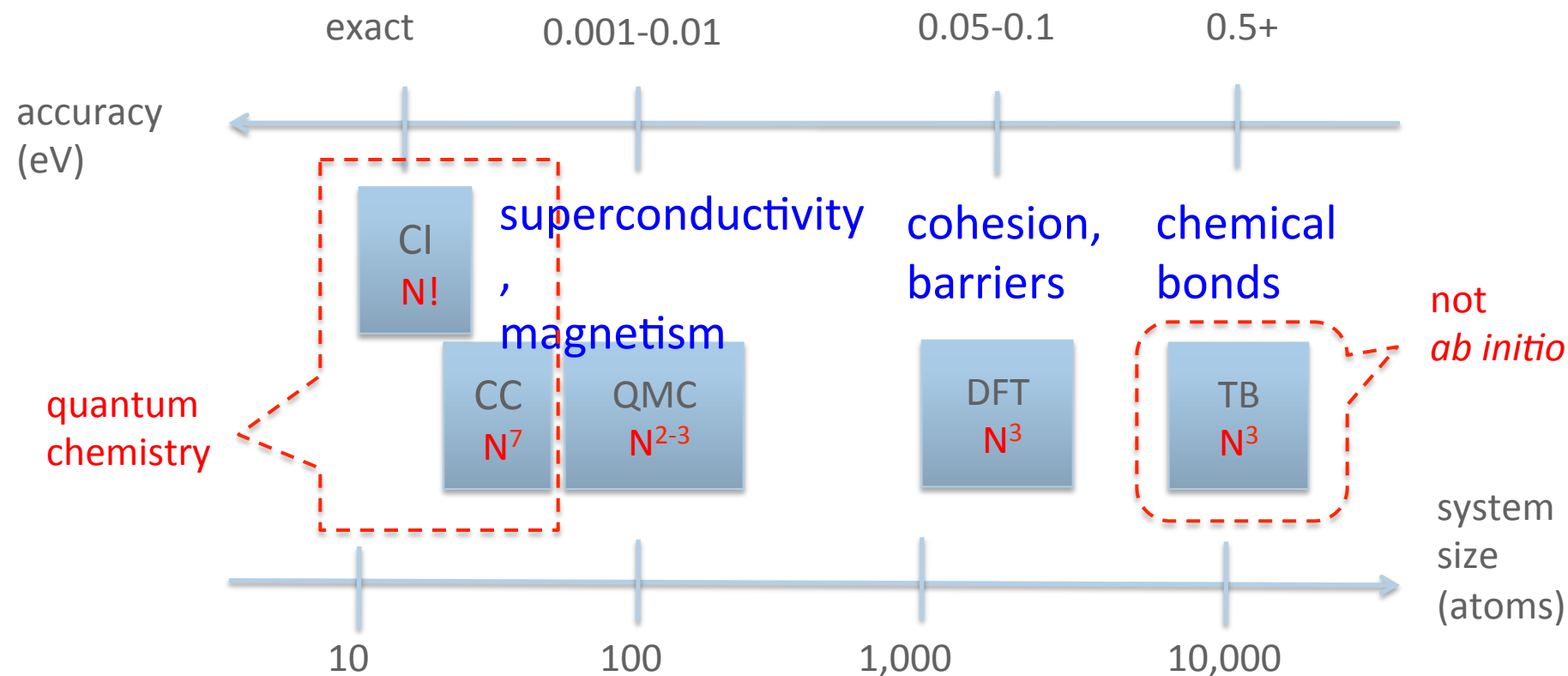
# Born-Oppenheimer Approximation



# Electronic Structure Methods

- Time scale: picosecond =  $10^{-12}$  seconds
- Length scale: 10 nm =  $10^{-8}$  meters

atom = 100 –  
1000 eV





# First-Principles Electronic Structure Theory

$$\left[ -\frac{\hbar^2}{2m} \vec{\nabla}^2 + V_{e-e}(\vec{r}, \{\vec{R}\}) + V_{e-l}(\vec{r}, \{\vec{R}\}) \right] \psi_q(\vec{r}) = E_q^{BO} \psi_q(\vec{r})$$





Only input:  
•  $N \alpha$   $R \alpha$ ,  $Z \alpha$



## Density Functional Theory

-  Mean-field cost, well developed codes, easy to learn, HPC.
-  Hard to improve, accuracy is unknown, fails for correlated problems!

## Quantum Chemistry Methods

-  Very accurate and robust, well developed codes.
-  High cost(> $N^5$ ), very hard for solids, complicated, no HPC.

## Quantum Monte Carlo

-  Accurate, improvable, trivially parallelizable, ok scaling ( $N^3$ )
-  High cost, human time intensive, small community, problems at high  $Z$ .



# Monte Carlo Simulations (Brief outline)





# Markov chain MC or Random Walk

- Markov chain is a random walk through phase space:

$$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow \dots$$

Here “ $s$ ” is the state of the system.

- ALL QMC is some type of Markov process. VMC is the simplest.
- The transition probability is  $P(s_n \rightarrow s_{n+1})$  a *stochastic matrix*

$$P(s \rightarrow s') \geq 0 \quad \sum_{s'} P(s \rightarrow s') = 1$$

- In a Markov chain, the distribution of  $s_{n+1}$  depends only on  $s_n$  (by definition). *A drunkard has no memory!*
- Let  $f_n(s)$  be the probability after “ $n$ ” steps. It evolves according to a “master equation.”

$$f_{n+1}(s') = \sum_s f_n(s) P(s \rightarrow s')$$

$$f_{n+1} = P f_n$$

- The stationary states are eigenfunctions of  $P$ :  $P f = e f$



# Ergodicity

In physics and thermodynamics, the ergodic hypothesis says that, over long periods of time, the time spent by a system in some region of the phase space of microstates ( $s$ ) with the same energy is proportional to the volume of this region, i.e., that all accessible microstates are equiprobable over a long period of time.

Statistically, for this random process, the time average of one sequence of events is the same as the ensemble average.

=> For a Markov Chain, as one increases the steps, there exists a positive probability measure at step  $n$  that is independent of probability distribution at initial step 0



# Metropolis algorithm

Three key concepts:

1. Sample by using an ergodic random walk.
2. Determine equilibrium state by using detailed balance.
3. Achieve detailed balance by using rejections.

**Detailed balance**  $\pi(s) P(s \rightarrow s') = \pi(s') P(s' \rightarrow s).$

*Rate balance from  $s$  to  $s'$ .*

Put  $\pi(s)$  into the master equation. (Or sum above Eq. on  $s$ .)

$$\sum_s \pi(s) P(s \rightarrow s') = \pi(s') \sum_s P(s' \rightarrow s) = \pi(s')$$

- Hence,  $\pi(s)$  is an eigenfunction.
- If  $P(s \rightarrow s')$  is ergodic,  $\pi(s)$  is unique steady state solution.



# Rejection Method

Metropolis achieves detailed balance by *rejecting* moves.

## *General Approach:*

1. Choose distribution to sample, e.g.,  $\pi(s) = \exp[-\beta H(s)]/Z$
2. Impose detailed balance on transition:  $K(s \rightarrow s') = K(s' \rightarrow s)$

where  $K(s \rightarrow s') = \pi(s) P(s \rightarrow s')$

*(probability of being at  $s$ ) \* (probability of going to  $s'$ ).*

3. Break up transition probability into sampling and acceptance:

$$P(s \rightarrow s') = T(s \rightarrow s') A(s \rightarrow s')$$

*(probability of generating  $s'$  from  $s$ ) \* (probability of accepting move)*

The optimal acceptance probability that gives detailed balance is:

$$A(s \rightarrow s') = \min\left[1, \frac{T(s' \rightarrow s) \pi(s')}{T(s \rightarrow s') \pi(s)}\right] = \min\left[1, \frac{\pi(s')}{\pi(s)}\right]$$

Normalization of  $\pi(s)$  is not needed or used!

If  $T$  is *constant*!



# The “Classic” Metropolis method

*Metropolis-Rosenbluth<sup>2</sup> -Teller<sup>2</sup> (1953) method is:*

- Move from  $s$  to  $s'$  with probability  $T(s \rightarrow s') = \text{constant}$
- Accept with move with probability:

$$A(s \rightarrow s') = \min [ 1 , \exp ( - (E(s') - E(s)) / k_B T ) ]$$

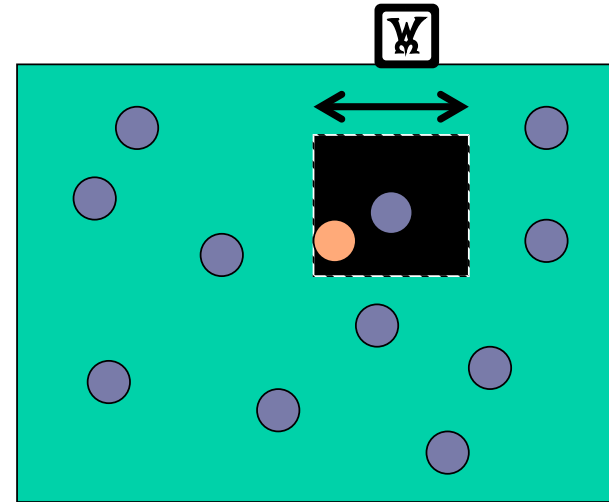
- Repeat many times
- Given ergodicity, the distribution of  $s$  will be the canonical distribution:  $\pi(s) = \exp(-E(s)/k_B T) / Z$
- Convergence is guaranteed but the rate is not!



# How to sample

$$S_{\text{new}} = S_{\text{old}} + \Delta \cdot (\text{sprng} - 0.5)$$

Uniform distribution in a cube of side “ $\Delta$ ”.



**Note:** It is more efficient to *move one particle at a time* because only the energy of that particle comes in and the *acceptance ratio will be larger*.

$$\begin{aligned} A(s \rightarrow s') &= \exp[-\beta(V(s') - V(s))] \\ &= \exp[-\beta \sum_{j \neq i} (v(r_i' - r_j) - v(r_i - r_j))] \end{aligned}$$

*For  $V$  with cut-off range, difference is local.*

# MONTE CARLO CODE

```

call initstate(s_old)
E_old = action(s_old)
LOOP{
    call sample(s_old,s_new,T_new,1)
    E_new = action(s_new)
    call sample(s_new,s_old,T_old,0)
    A=exp(-E_new+E_old) T_old/T_new
    if(A.gt.sprng()) {
        s_old=s_new
        E_old=E_new
        naccept=naccept+1}
    call averages(s_old)
}

```

← Initialize the state

← Sample snew

← Trial action

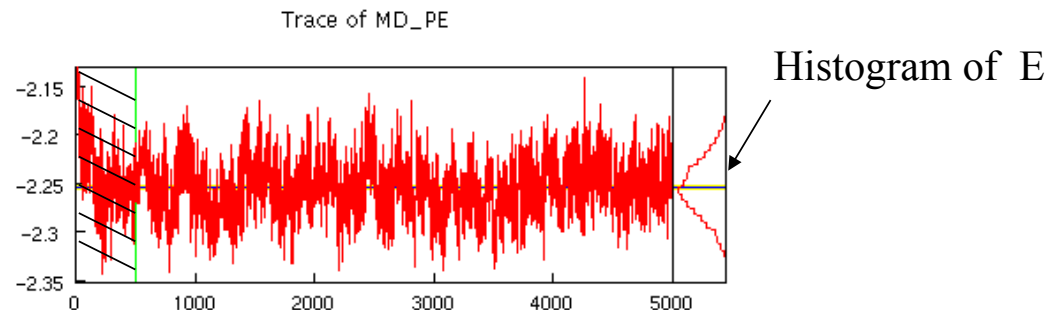
← Find prob. of going backward

← Acceptance prob.

← Accept the move  
Collect statistics

# Estimated Errors

- **In what sense do we calculate exact properties?** Answer: if we average long enough the error goes to zero, the errors of the simulation are controlled.
- Next, how accurate is the estimate of the exact value?
  - Simulation results without error bars are only suggestive.
    - Without error bars one has no idea of its significance.
    - You should **understand formulas and be able to make an “eye-ball” estimate**.
- **Error bar:** the *estimated error* in the *estimated mean*.
  - Error estimates based on Gauss' *Central Limit Theorem*.
  - **Average** of statistical processes has *normal* (Gaussian) distribution.
  - **Error bars:** square root of the variance of the distribution divided by the number of *uncorrelated* steps.





# Quantum Monte Carlo Simulations



# Quantum Monte Carlo

- We need to use simulation techniques to “solve” many-body quantum problems just as you need them classically.
- Both the wavefunction and expectation values are determined by the simulations. Correlation built in from the start.
- Based on Feynman’s imaginary time path integrals.
- QMC gives most accurate method for general quantum many-body systems.
- QMC determined electronic energy is the standard for approximate LDA calculations. (but fermion sign problem!)
- Path Integral Methods provide a exact way to include effects of ionic zero point motion (include all anharmonic effects)
- A variety of stochastic QMC methods:
  - **Variational Monte Carlo VMC ( $T=0$ )**
  - **Projector Monte Carlo ( $T=0$ )**
    - **Diffusion MC (DMC)**
    - **Reptation MC (RQMC)**
  - **Path Integral Monte Carlo (PIMC) ( $T>0$ )**
  - **Coupled Electron-Ion Monte Carlo (CEIMC)**



# Quantum Monte Carlo Simulations Variational Monte Carlo (VMC)



# Variational Monte Carlo (VMC)

- Variational Principle. Given an appropriate trial function:
  - Continuous
  - Proper symmetry
  - Normalizable
  - **Finite variance**
- Quantum chemistry uses a product of single particle functions
- With MC we can use any “computable” function.

- Sample  $R$  from  $|\psi|^2$  using MCMC.
  - Take average of local energy:
  - Optimize  $\psi$  to get the best upper bound
  - Error in energy is 2<sup>nd</sup> order

- Better wavefunction, lower variance!  
**“Zero variance” principle.** (non-classical)

$$E_V = \frac{\int dR \langle \psi | H | \psi \rangle}{\int dR \langle \psi \psi \rangle} \geq E_0$$

$$\sigma^2 = \frac{\int dR \langle \psi | H^2 | \psi \rangle}{\int dR \langle \psi \psi \rangle} - E_V^2$$

$$E_L(R) = \Re [\psi^{-1}(R) H \psi(R)]$$

$$E_V = \langle E_L(R) \rangle_{\psi^2} \geq E_0$$



## VARIATIONAL MONTE CARLO CODE

```

call initstate (s_old)
p_old = psi2 (s_old)
LOOP {
  call sample (s_old,s_new,T_new,1)
  p_new = psi2 (s_new)
  call sample (s_new,s_old,T_old,0)
  A = (p_new/T_new)/(p_old/T_old)
  if(A > rand () ) {
    s_old=s_new
    p_old=p_new
    naccept = naccept +1}
  call averages (s_old)
}

```

← Initialize the state  
 Evaluate  $\psi_{\text{trial}}$   
 ← Sample new state  
 Evaluate  $\psi_{\text{trial}}$   
 ← Find transition prob.  
 for going backward  
 ← Acceptance prob.  
 ← Accept the move  
 ← Collect statistics

# Quantum Monte Carlo Simulations

## Diffusion Monte Carlo (DMC)



# Projector Monte Carlo

(variants: Green's function MC, Diffusion MC, Reptation MC)

- Project single state using the Hamiltonian

$$\phi(t) = e^{-(H-E_T)t} \phi(0)$$

- We show that this is a diffusion + branching operator. Maybe we can interpret as a probability. **But is this a probability?**
- **Yes!** for bosons since ground state can be made real and non-negative.
- **But** all excited states must have sign changes. This is the “sign problem.”
- For efficiency we do “importance sampling.”
- Avoid sign problem with the fixed-node method.



# Diffusion Monte Carlo

- How do we analyze this operator?

$$\psi(R, t) = e^{-(H-E_T)t} \psi(R, 0)$$

- Expand into exact eigenstates of H.

$$H\phi_\alpha = E_\alpha \phi_\alpha$$

$$\psi(R, 0) = \sum_{\alpha} \phi_{\alpha}(R) \langle \phi_{\alpha} | \psi(0) \rangle$$

- Then the evolution is simple in this basis.

$$\psi(R, t) = \sum_{\alpha} \phi_{\alpha}(R) e^{-t(E_{\alpha}-E_T)} \langle \phi_{\alpha} | \psi(0) \rangle$$

- Long time limit is lowest energy state that overlaps with the initial state, usually the ground state.

$$\lim_{t \rightarrow \infty} \psi(R, t) = \phi_0(R) e^{-t(E_0-E_T)} \langle \phi_0 | \psi(0) \rangle$$

$$E_0 \approx E_T \Rightarrow \text{normalization fixed}$$





# Importance Sampling

*Kalos 1970, Ceperley 1979*

- Why should we sample the wavefunction? The physically correct pdf is  $|\Phi|^2$ .
- Importance sample (multiply) by trial wave function.

$$f(R, t) \equiv \psi_T(R) \phi(R, t) \quad \lim_{t \rightarrow \infty} f(R, t) \equiv \psi_T(R) \phi_0(R)$$

$$-\frac{\partial f(R, t)}{\partial t} = \psi_T(R) H \left[ f(R, t) / \psi_T(R) \right] \quad \text{Commute } \Psi \text{ through } H$$

$$-\frac{\partial f(R, t)}{\partial t} = -\frac{1}{2} \nabla^2 f - \nabla \left( f \nabla \ln \psi_T(R) \right) + \left( \psi_T^{-1} H \psi_T \right) f(R, t)$$

Evolution = diffusion + drift + branching

- Use accept/reject step for more accurate evolution.  
make acceptance ratio > 99% . Determines time step.

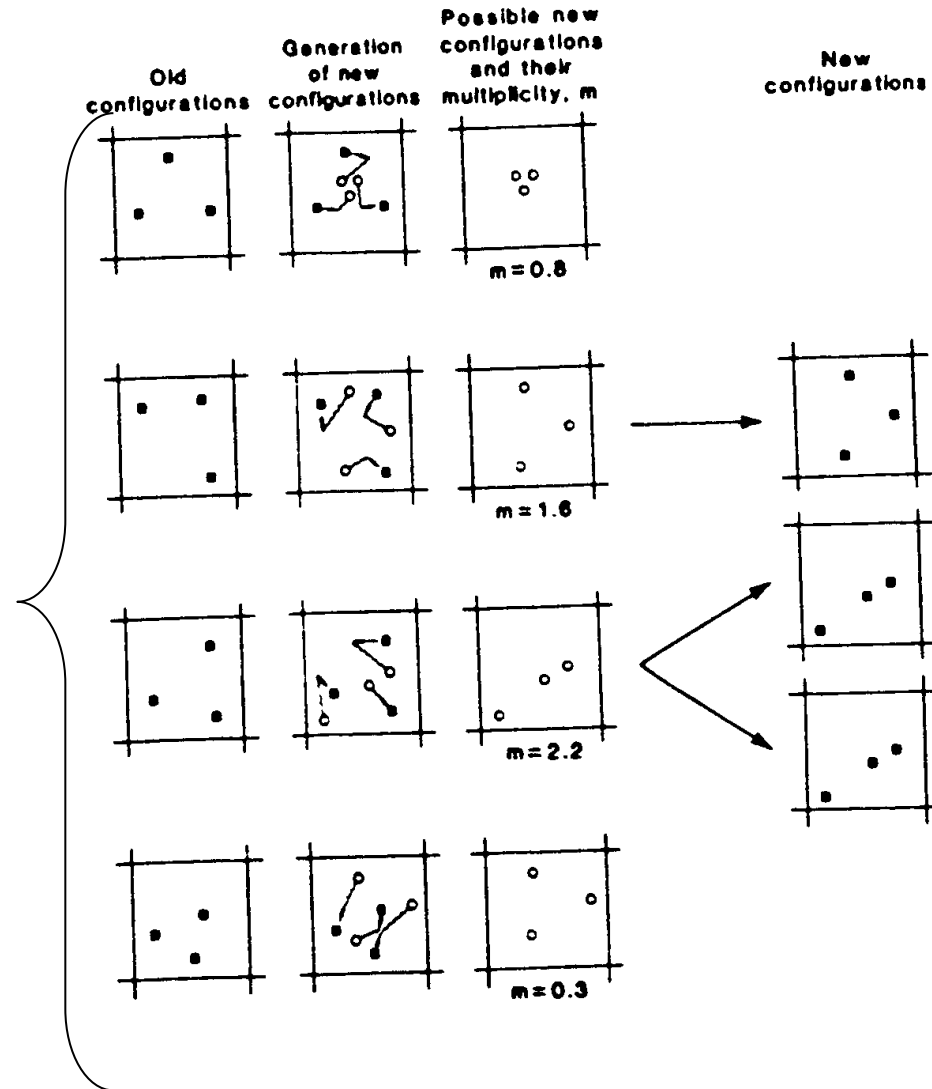


# Schematic of DMC

Ensemble evolves according to

- Diffusion
- Drift
- branching

**ensemble**



# DIFFUSION MONTE CARLO CODE

```

call initstate(s_old)
psi_old = psi(s_old)
d_old = drift(s_old)
LOOP {
  LOOP {
    call sample(s_old,s_new,T_new,d_old,1)
    psi_new = psi(s_new)
    if (psi_new * psi_old < 0 ) {
      weight = 0
    } else {
      d_old = drift(s_old)
      call sample(s_new,s_old,T_old,d_new,0)
      A = (p_new/T_new)/(p_old/T_old)
      if(A > rand() ) {
        s_old=s_new
        p_old=p_new
        naccept = naccept +1 } }
      weight *= exp(- tau * local_energy(s_old) ) }
    call reweight(s_old)
    call averages(s_old)
  }
}

```

*Initialize the ensemble of states*  
*Evaluate psi\_trial*  
*Evaluate grad psi\_trial*  
*Loop over steps*  
*Loop over walkers*  
*Sample new state from drifted Gaussian*  
*Evaluate psi\_trial*  
*Check node crossing*  
*Kill walker if it crosses a node of psi\_trial*  
*Evaluate grad psi\_trial*  
*Find transition prob. for going backward*  
*Acceptance prob.*  
*Accept the move*  
*Update weight*  
*Reweight ensemble*  
*Collect statistics*

# Quantum Monte Carlo Simulations

## Trial Wavefunction



# What do we use for the trial function?

- **Formal requirements** antisymmetry, continuity, finite variance.
- **Mean field approaches** Slater determinant,
- **Expand in basis sets** e.g. multideterminants, CI.
- **Local energy argument**
  - Control singularity at small  $r$ , or  $r_{12}$ . Leads to cusp condition.
  - Behavior at large  $r$ , plasmons, van der Waals (dispersion) interaction
- **Feynman-Kacs formula** gives connection to local energy
  - Jastrow, Backflow, three-body interactions



## The many-body trial wavefunction

### Correlation (Jastrow)

$$J_1 = \sum_i^N \sum_l^{N_{ions}} u_1(|r_i - r_l|)$$

$$J_2 = \sum_{i \neq j}^N u_2(|r_i - r_j|)$$

$$\Psi_T(R) = J(R) \Psi_{AS}(R) = e^{J_1 + J_2 + \dots} \sum_k^M C_k D_k^\uparrow(\phi) D_k^\downarrow(\phi)$$

### Anti-symmetric function (Pauli principle)

$$D_k^\sigma = \begin{vmatrix} \phi_1(r_1) & \cdots & \phi_1(r_{N^\sigma}) \\ \vdots & \ddots & \vdots \\ \phi_{N^\sigma}(r_1) & \cdots & \phi_{N^\sigma}(r_{N^\sigma}) \end{vmatrix}$$

Single-particle orbitals

$$\phi_i = \sum_{l=1}^{N_b} C_l^i \Phi_l$$

Basis sets: molecular orbitals,  
plane-wave, grid-based orbitals...

$$\Phi_l$$

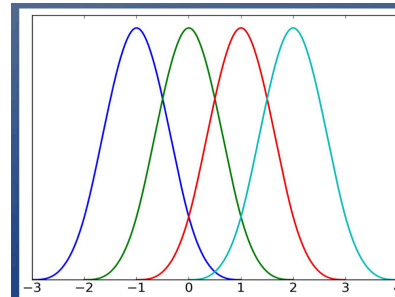
# Single particle orbitals

- The most common source for single particle orbitals for solid state calculations is from a DFT code
- These wavefunctions are typically represented in a plane wave basis:
$$\phi_i(\vec{r}) = \sum_j c_{ij} e^{i\vec{k}_j \cdot \vec{r}}$$
- Each single particle move requires evaluating every orbital at a new position
  - In a plane wave basis set, this means all N plane waves must be evaluated for every orbital!

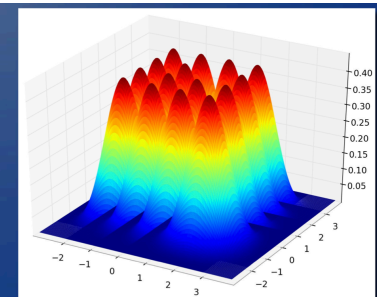


# Single particle orbitals

- Plane wave representations are inefficient
- A real space basis is much more efficient
  - Extensive testing shows that 3D b-splines are a good choice for efficiency
  - Only 64 basis elements are nonzero at each point (versus thousands or more for plane waves)



1D



2D

- $B_{ijk}(x,y,z) = a_i(x)b_j(y)c_k(z)$
- Efficient routines exist to convert from a plane wave basis to b-splines with an arbitrary real space spacing using the FFT





David Ceperley  
Norman Tubman  
Raymond Clay



Paul Kent  
Jaron Krogel



Anouar Benali

# QMCPACK

## Quantum Monte Carlo Simulation Package



Luke Shulenburger



Miguel Morales

Jeongnim Kim (Intel)  
Ken Esler (StoneRidge)  
Jeremy McMinis (Private)



# QMCPACK

Today, developed by a large number of developers, interested parties

Builds in ~all common computing environments (HPC, clusters, workstations)

C++, Object oriented, extensible, very efficient

MPI Parallelized

Fully OpenMP threaded (important for e.g. large solids by reducing memory usage + increasing efficiency)

GPU CUDA version ( currently single determinant, splined wavefunctions )

Standardized XML inputs & HDF5 input/output for large data



## B-splines and Memory

- QMCPACK uses a regularly spaced grid of points to calculate the 3d b-splines
  - Extremely efficient lookup in memory and streaming of many wavefunctions to CPU
  - Inefficient in terms of total memory used
    - All wavefunctions may be small in a region of space (for instance if vacuum is included in the calculation)
    - Wavefunctions typically vary most rapidly near nuclei
    - Some single-particle wavefunctions may be small in some regions but large in others



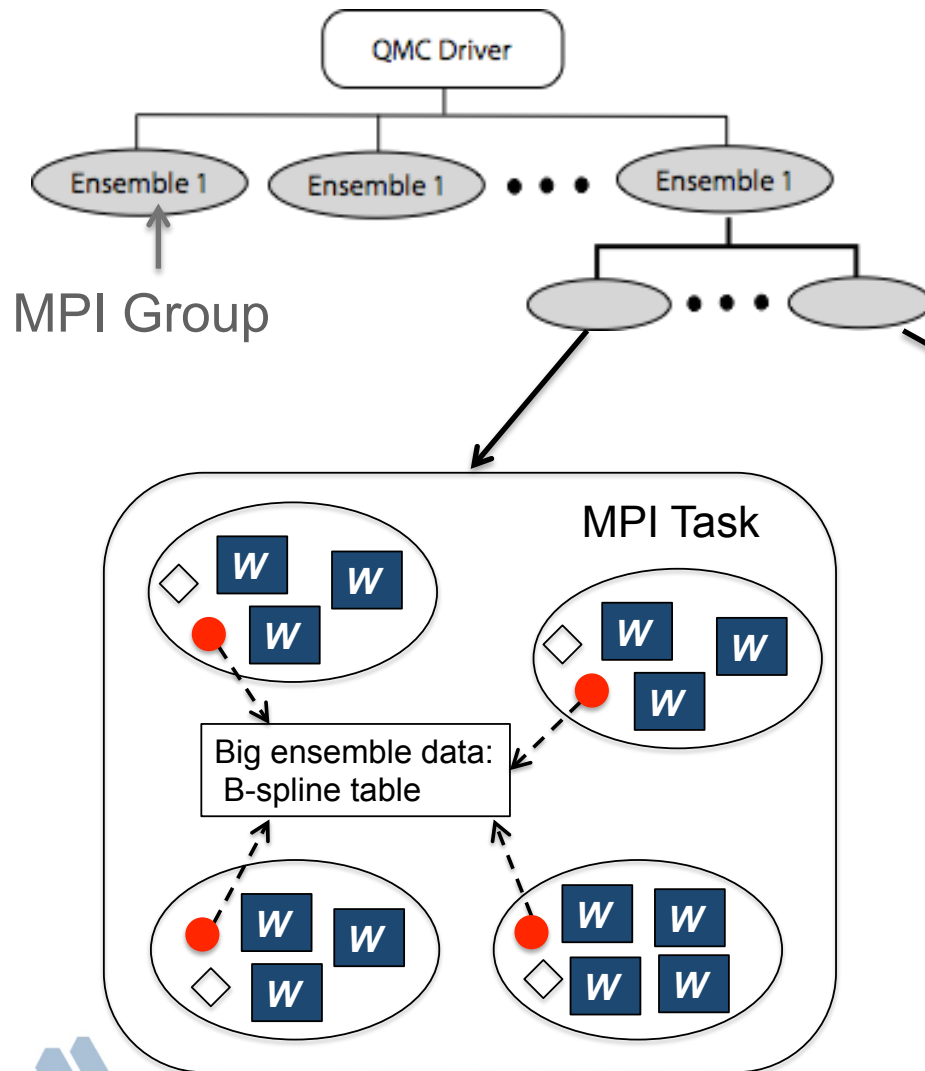
# QMCPACK Hybrid Parallelism and memory

- Moves for a given walker are evaluated by a single processing unit (CPU core)
  - So every core has to be able to quickly evaluate all single particle wavefunctions at all points in space
  - Modern supercomputers typically have 10s of cores per node
  - The single particle wavefunctions are never changed during the calculations
  - Share them between all of the cores on a node
- QMCPACK uses a combination of MPI and OpenMP for parallelization
  - OpenMP on a node makes sharing large static data for the wavefunction easy
- Reduces required memory by nearly a factor of the number of cores per node

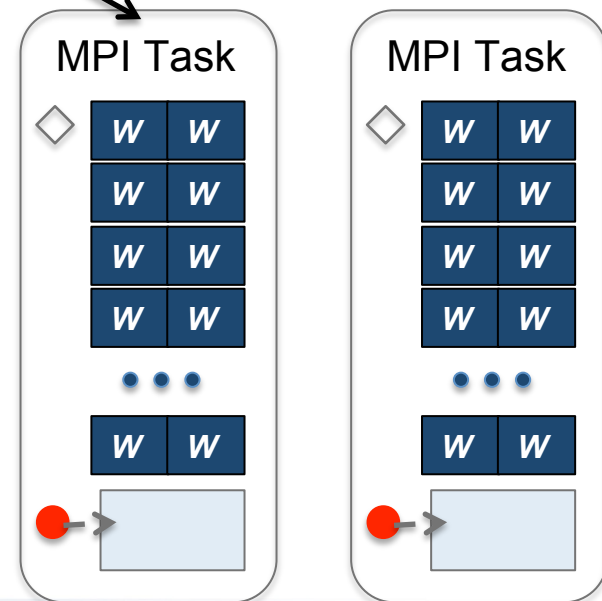


# QMCPACK on Clusters of SMP

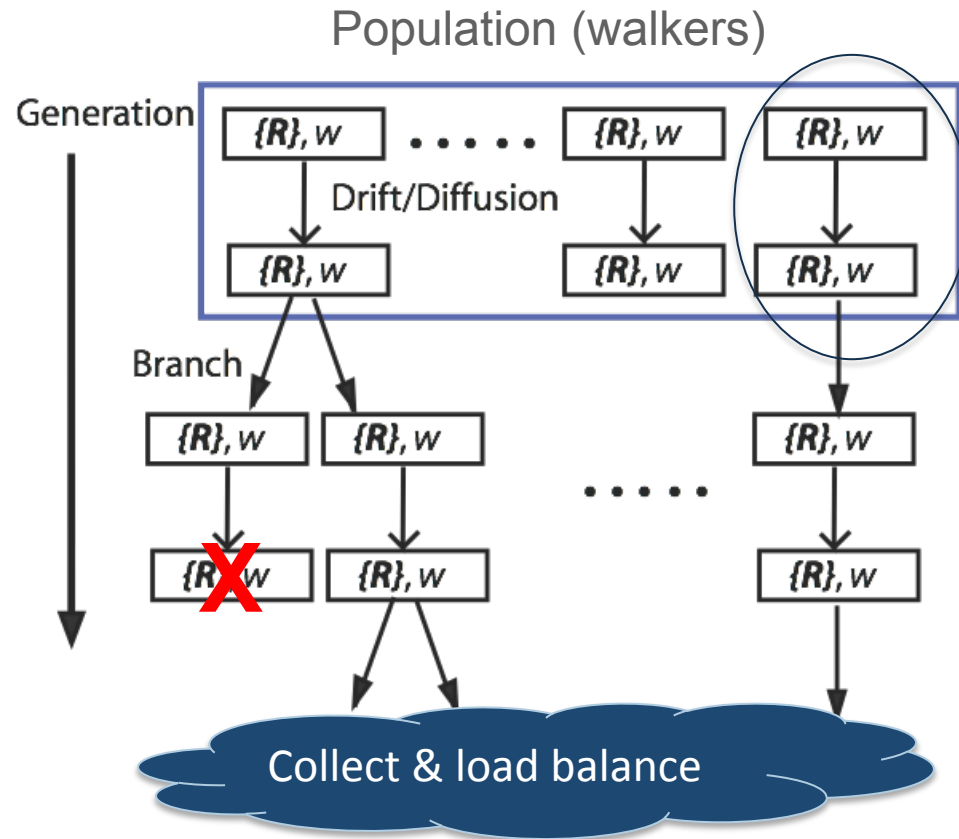
MPI+X Model for QMC, X=(OpenMP,CUDA, threads,?)



For performance: make the most of shared memory domain and SIMD



# QMCPACK: computational view



Make a move

$$\mathbf{R}' = \mathbf{R} + \tau \nabla \ln \Psi_T(\mathbf{R}) + \chi$$

Random

“Quantum Force”

Accept/reject a move

$$\frac{|\Psi_T(\mathbf{R}')|^2}{|\Psi_T(\mathbf{R})|^2} \frac{G_d(\mathbf{R} \rightarrow \mathbf{R}'; \tau)}{G_d(\mathbf{R}' \rightarrow \mathbf{R}; \tau)}$$

Branch with the weight

$$\exp^{-\tau[(E_L(\mathbf{R}) + E_L(\mathbf{R}'))/2 - \tilde{E}_T]}$$

- **Computationally Intensive:** Quantum Force, Ratio, Local Energy (Opportunity to be “thread nested”)
- Communication light
- Ample parallel opportunities : configurations, k-point, walkers

# (MPI+X)/C++ has worked (so far)

- Simple but effective for QMC
- Productive solution for developers (physicists)
  - Abstraction for physics concepts, methods and algorithms
  - Extensible and efficient
- Object-oriented and generic programming
  - Allow separation of science from coding
  - Lazy implementations and optimizations
- Efficient memory and thread management
  - Oversubscription on Intel & IBM BGQ increases performance
- Portable and scalable from a laptop to HPC
- Optimal use of current multi/many-core SMP architectures
- Exploit steady improvement of HPC ecosystem
  - Compilers, open standards, systems, numerical libraries, I/O



# Performance on BGQ







## Mira: next-generation supercomputer

48 racks  
1,024 nodes per rack  
16 cores per node  
64 threads per node  
16GB memory/node  
1.6GHz 16-way core processor  
240 GB/s, 35 PB storage

786k cores  
8.15 PF/s



# BGQ - High Performance features

## Quad FPU (QPU)

DMA unit

### List-based prefetcher

TM (Transactional Memory)

SE (Speculative Execution)

Wakeup-Unit

Scalable Atomic Operations



Instruction Extensions (QPX) to PowerISA  
4-wide double precision FPU SIMD (BG/L,P are 2-wide) usable as:

scalar FPU

4-wide FPU SIMD

2-wide complex arithmetic SIMD

Attached to AXU port of A2 core – A2 issues one instruction/cycle to AXU

8 concurrent floating point operations (FMA) + load +store

- 6 stage pipeline

Permute instructions to reorganize vector data  
supports a multitude of data  
alignments

4R/2W register file

32x32 bytes per thread

32B (256 bits) data path to/from L1 cache

## Intrinsic:

TYPE: vector4double A;

Loads and stores

Unary operations

Binary operations

Multiply-add operations

Special functions



# Profiling

## HPM Profile with original version of QMCPACK

### System:

- Ar Solid – 32 atoms – 256 electrons – B-splines representation of WF (1.9Gb) :
- 256 nodes – 32 threads – 2 Walkers per thread
- **Total run time: 53min40**

Percentage of peak= 6.55%

```
27.644.290.379.027    All XU Instruction
22.786.190.220.714    All AXU Instruction
43.043.218.198.088    FP Operations Group 1
```

Derived metrics for code block "mpiAll" averaged over process(es) on node <0,0,0,0,0>:

Instruction mix: **FPU = 45.18 %**, **FXU = 54.82 %**

Instructions per cycle completed per core = **0.6138**

Per cent of max issue rate per core = 33.65 %

Total weighted GFlops for this node = 13.412

Loads that hit in **L1 d-cache = 94.03 %**

**L1P buffer = 5.36 %**

L2 cache = 0.35 %

**DDR = 0.26 %**

DDR traffic for the node: ld = 1.508, st = 0.540, total = 2.049  
(Bytes/cycle)



# Profiling

System:

- Ar Solid – 32 atoms – 256 electrons – B-splines representation of WF (1.9Gb) :
- 256 nodes – 32 threads – 2 Walkers per thread

Profile with original version of QMCPACK

Flat profile:

**Total run time: 53min40**

Each sample counts as 0.01 seconds.

% cumulative self self total

time seconds seconds calls Ts/call Ts/call name

**56.95** 58369.57 58369.57

**.eval\_multi\_UBspline\_3d\_z\_vgh**

**14.02** 72738.82 14369.25

**.eval\_multi\_UBspline\_3d\_z**

2.11 77918.51 2161.01

SymmetricDTD

1.70 79663.07 1744.56

EinsplineSetExtended::evaluate

**Evaluation of spline, gradient and  
hessian coefficients (complex)**

**Evaluation of spline  
coefficients (complex)**

71% of the application time spent in the Spline evaluation of the Wave Function



# Einspline

Eval\_z\_vgh:

Evaluation of spline coefficients, gradient and hessian (complex)

```
for (int i=0; i<4; i++)
    for (int j=0; j<4; j++)
        for (int k=0; k<4; k++) {
//missing code (definition of abc
and loading arithmetic pointer coefs)
    for (int n=0; num_splines; n++) {
        vals[n]      += abc    *coefs[n];
        grads[3*n+0] += dabc[0]*coefs[n];
        grads[3*n+1] += dabc[1]*coefs[n];
        grads[3*n+2] += dabc[2]*coefs[n];
        hess [9*n+0] += d2abc[0]*coefs[n];
        hess [9*n+1] += d2abc[1]*coefs[n];
        hess [9*n+2] += d2abc[2]*coefs[n];
        hess [9*n+4] += d2abc[3]*coefs[n];
        hess [9*n+5] += d2abc[4]*coefs[n];
        hess [9*n+8] += d2abc[5]*coefs[n];
    }
}
```

## Alternative

- Inner loop to Outer loop
- Unroll (j) and (k) loops
- Reformulate the arithmetic problem
- Load data from non contiguous



```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;  
    for (int n=0; n<num_splines; n++, coefs++) {  
        for (int i=0; i<4; i++) {
```

```
        complex_double coef00 = coefs[i*xs];  
        complex_double coef11 = coefs[i*xs + zs];  
        complex_double coef22 = coefs[i*xs + 2*zs];  
        complex_double coef33 = coefs[i*xs + 3*zs];
```

```
        val = a[i] { (coef00*c[0]+coef01*c[1]+coef02*c[2]+coef03*c[3]) *b[0]  
                    + (coef10*c[0]+coef11*c[1]+coef12*c[2]+coef13*c[3]) *b[1]  
                    + (coef20*c[0]+coef21*c[1]+coef22*c[2]+coef23*c[3]) *b[2]  
                    + (coef30*c[0]+coef31*c[1]+coef32*c[2]+coef33*c[3]) *b[3]  
                    }
```

Reminder:

vector4double : {doubles,double,double,double}

Complex\_double : {double,double}



```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
for (int n=0; n<num_splines; n++, coefs++) {
    for (int i=0; i<4; i++) {
```

```
    val = a[i]{(coef00*c[0]+coef01*c[1]+coef02*c[2]+coef03*c[3])*b[0]
```

```
    val = a[i]{
        (real(coef00)*c[0]+real(coef01)*c[1]+real(coef02)*c[2]+real(coef03)*c[3]
        imag(coef00)*c[0]+imag(coef01)*c[1]+imag(coef02)*c[2]+imag(coef03)*c[3])*b[0]
        //code omitted
    }
```

```
    val = a[i]{
        (real(coef00)*c[0] + real(coef02)*c[2]
        imag(coef00)*c[0] + imag(coef02)*c[2]
        real(coef01)*c[1] + real(coef03)*c[3]
        imag(coef01)*c[1] + imag(coef03)*c[3]) *b[0]
```

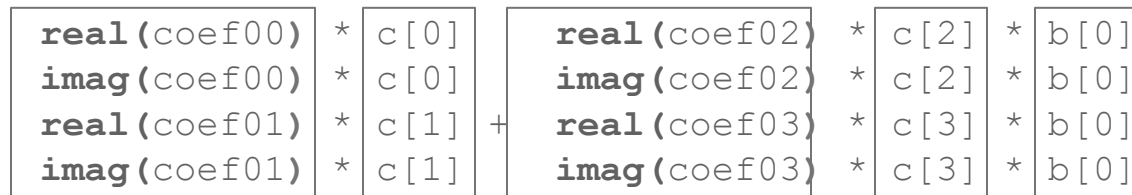


```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
for (int n=0; n<num_splines; n++, coefs++) {
    for (int i=0; i<4; i++) {
```

```
    val = a[i]{(coef00*c[0]+coef01*c[1]+coef02*c[2]+coef03*c[3])*b[0]
```

```
    val = a[i]{
        (real(coef00)*c[0]+real(coef01)*c[1]+real(coef02)*c[2]+real(coef03)*c[3]
        imag(coef00)*c[0]+imag(coef01)*c[1]+imag(coef02)*c[2]+imag(coef03)*c[3])*b[0]
        //code omitted
    }
```

```
    val = a[i]{
```



```
vector4double b_0=vec_splat(b,0);
```

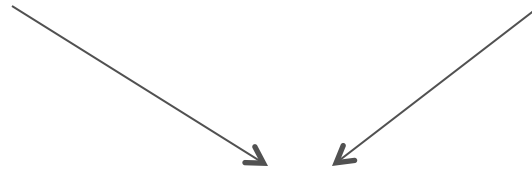




```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
for (int n=0; n<num_splines; n++, coefs++) {
    for (int i=0; i<4; i++) {

        val = a[i]{
```

$$\begin{array}{|l|} \hline \mathbf{real(coef00)} \\ \hline \mathbf{imag(coef00)} \\ \hline \mathbf{real(coef01)} \\ \hline \mathbf{imag(coef01)} \\ \hline \end{array} * \begin{array}{|l|} \hline c[0] \\ \hline c[0] \\ \hline c[1] \\ \hline c[1] \\ \hline \end{array} + \begin{array}{|l|} \hline \mathbf{real(coef02)} \\ \hline \mathbf{imag(coef02)} \\ \hline \mathbf{real(coef03)} \\ \hline \mathbf{imag(coef03)} \\ \hline \end{array} * \begin{array}{|l|} \hline c[2] \\ \hline c[2] \\ \hline c[3] \\ \hline c[3] \\ \hline \end{array} * \begin{array}{|l|} \hline b[0] \\ \hline b[0] \\ \hline b[0] \\ \hline b[0] \\ \hline \end{array}$$



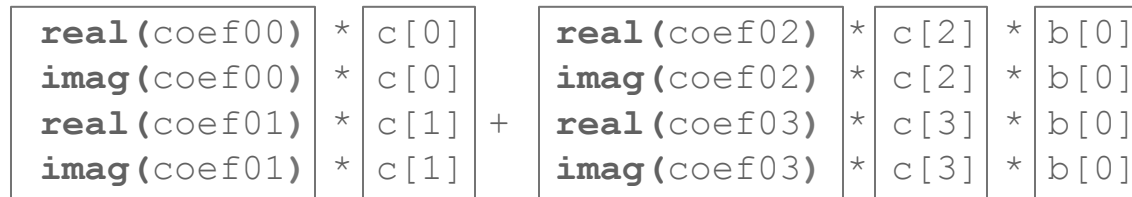
```
vector4double Coef0_0_0=vec_ld2(0, (double*) (coefs+(i*xs)));
vector4double Coef0_0_1=vec_ld2(0, (double*) (coefs+(i*xs+zs)));
```

```
vector4double Coef0_1_0=vec_sldw(Coef0_0_0,Coef0_0_1,2);
```



```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
for (int n=0; n<num_splines; n++, coefs++) {
    for (int i=0; i<4; i++) {

        val = a[i]{
```



```
vector4double mantissa_1 = vec_gpci(0011);
vector4double mantissa_2 = vec_gpci(02233);
```

```
vector4double splatc_0_1=vec_perm(c,c,mantissa_1);
vector4double splatc_0_2=vec_perm(c,c,mantissa_2);
```

```
vector4double Coef01=vec_add(vec_mul(Coef0_1_0,splatc_0_1),vec_mul(Coef0_2_0,splatc_0_2));
```



# Einspline

Eval\_z\_vgh:

Evaluation of spline coefficients,

Gradient and Hessian (complex)

Number of Cycles = 1879207

503.975 All XU Instruction

619.513 All AXU Instruction

1.141.770 FP Operations Group 1

Derived metrics for code block

"**Original**" averaged over process(es)

on node <0,0,0,0,0>:

Instruction mix: FPU = 55.14 %,

FXU = 44.86 %

Instructions per cycle completed

per core = **0.5940**

Per cent of max issue rate per core = 59.40 %

Total weighted GFlops for this node = 0.966

Loads that hit in L1 d-cache = 93.44 %

L1P buffer = **5.52** %

L2 cache = **0.83** %

DDR = **0.21** %

DDR traffic for the node: ld = 0.009,

st = 0.016, total = 0.025 (Bytes/cycle)

Number Of Cycles = 246752

52.395 All XU Instruction

32.802 All AXU Instruction

160.874 FP Operations Group 1

Derived metrics for code block

"**Benali QPX**" averaged over process(es)

on node <0,0,0,0,0>:

Instruction mix: FPU = **38.50** %,

FXU = **61.50** %

Instructions per cycle completed

per core = **0.3115**

Per cent of max issue rate per core = 31.15 %

Total weighted GFlops for this node = 0.941

Loads that hit in L1 d-cache = **75.10** %

L1P buffer = 1.00 %

L2 cache = 20.06 %

DDR = 3.84 %

DDR traffic for the node: ld = 0.068, st =

0.121, total = 0.190 (Bytes/cycle)

**Speedup compared to original: 7.62**



# Profiling

System:

- Ar Solid – 32 atoms – 256 electrons – Bsplines WF (1.9Gb) :
- 256 nodes – 32 threads – 2 Walkers per thread

Profile with QPX and Prefetch

Flat profile:

**Total run time: 20min03**

Each sample counts as 0.01 seconds.

% cumulative self

time seconds seconds

**14.08** 5380.43 5380.43

**8.25** 12270.83 3152.52

5.68 14441.45 2170.62

4.85 16292.97 1851.52

**.eval\_multi\_UBspline\_3d\_z\_vgh**

**.eval\_multi\_UBspline\_3d\_z**

.SymmetricDTD

EinsplineSetExtended::evaluate

Profile with Original Algorithm

Flat profile:

**Total run time: 53min40**

Each sample counts as 0.01 seconds.

% cumulative self

time seconds seconds

**56.95** 58369.57 58369.5

**14.02** 72738.82 14369.25

2.11 77918.51 2161.01

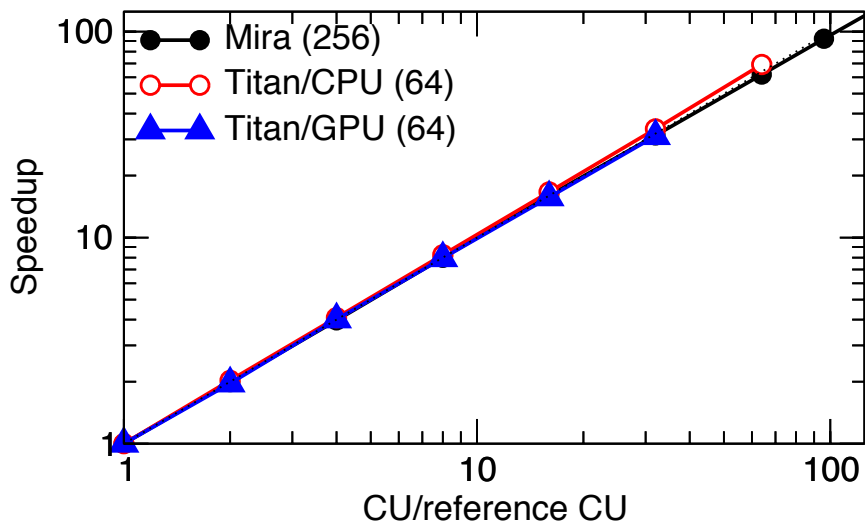
1.70 79663.07 1744.5

**Total run time Speedup of 2.68 times**



- DMC “weak” scaling is almost perfect , > 90% efficiency
  - Limited by collectives for  $E_T, N_p^w - < N^w >$
- 1 MPI to 1 NUMA or accelerator mapping: Gain in Collectives; easy to balance walkers per node

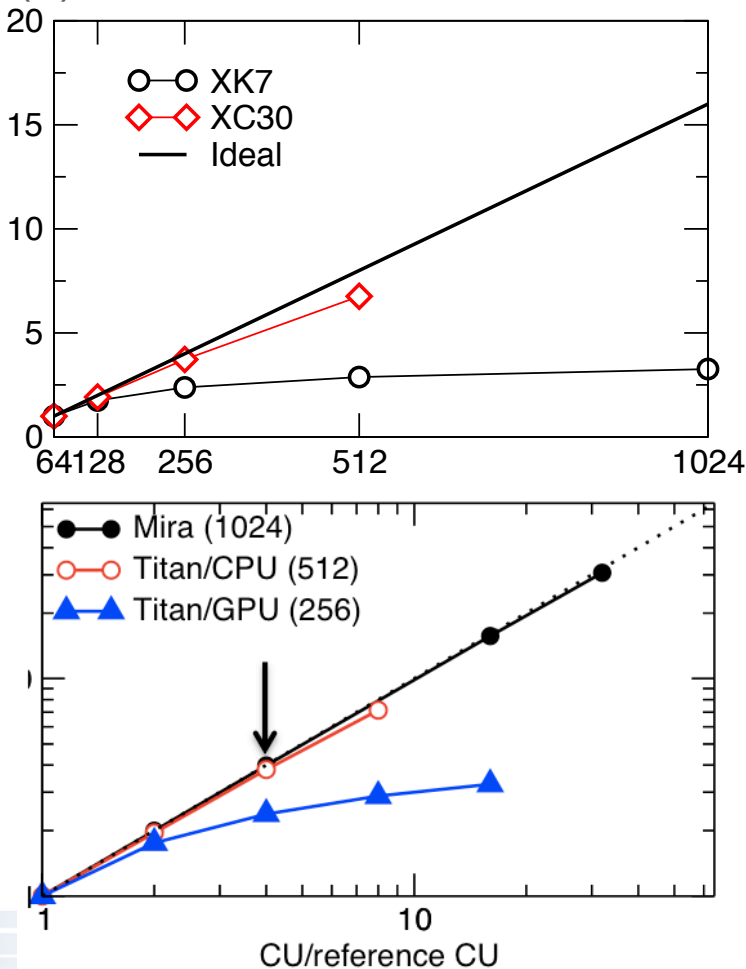
(a) Fixed walker per node



- (Nodes) for the reference Compute Unit
  - Mira: 16-core Blue Gene/Q
  - Titan: 16-core (CPU) and 1 GPU XK7

Optimize productivity

(b) Fixed total walker



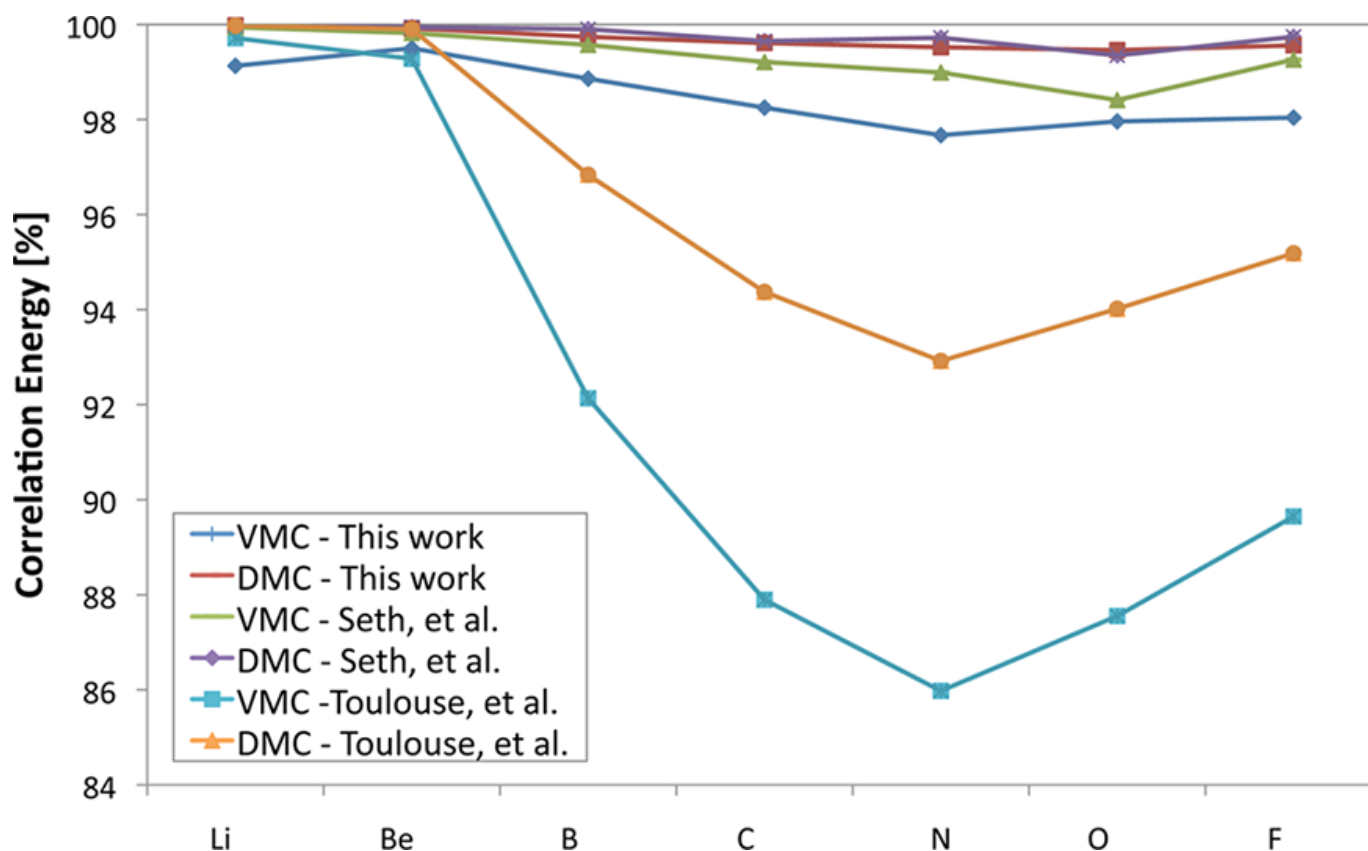
# Applications



# Benchmarking the accuracy of QMC for molecules

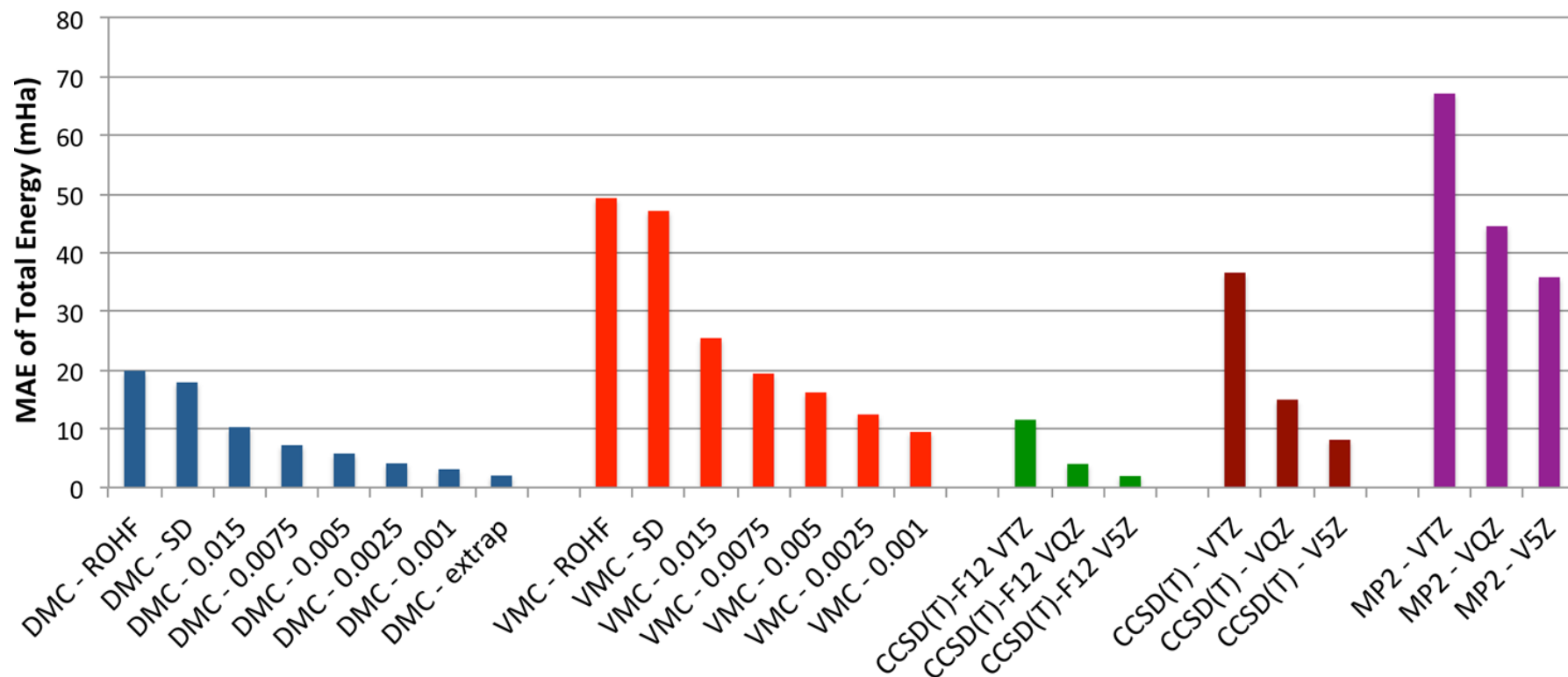
“Multideterminant Wave Functions in Quantum Monte Carlo”, M. A. Morales, ..., G.E.Scuseria. JCTC **8** 2182 (2012) <http://dx.doi.org/10.1021/ct3003404>

High accuracy achievable for atoms:



# Benchmarking the accuracy of QMC for molecules

Testing accuracy for G1 test set ( $\text{C}_2\text{H}_2$ , CN,  $\text{H}_2\text{O}$ , NaCl,  $\text{SiH}_4$ , SiO...), going beyond single determinant “standard recipe”. MAE of 0.8kcal/mol achieved for atomization energies, i.e. Chemical accuracy.



See also Nemec et al. JCP **132** 034111(2010)

<http://dx.doi.org/10.1063/1.3288054> all electron single determinant tests for same test set (CASINO code).

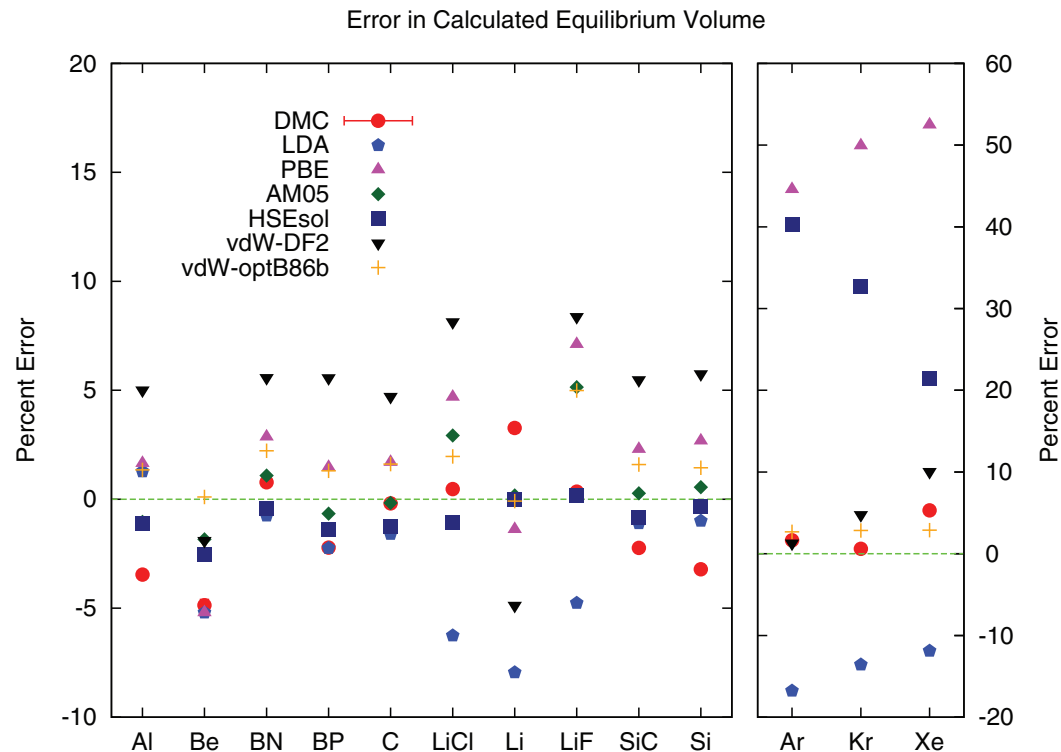




# Benchmarking the accuracy of QMC for solids

Analysis of structural properties ( $V$ ,  $B_0$ ) of ionic, metallic, covalent and van der Waals solids using “standard recipe” single determinant Slater-Jastrow pseudopotentials. Careful convergence of calculations. Finds high accuracy over whole set of solids.

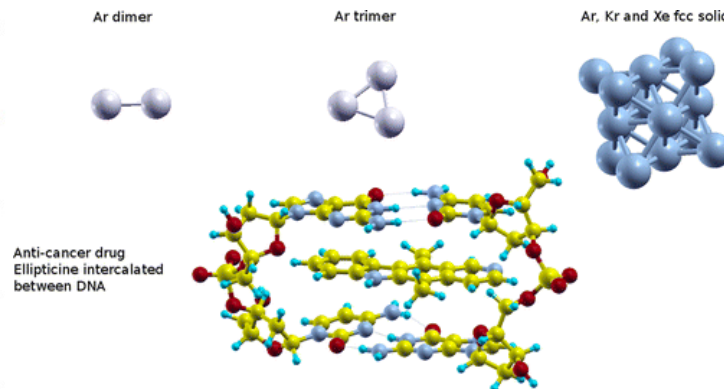
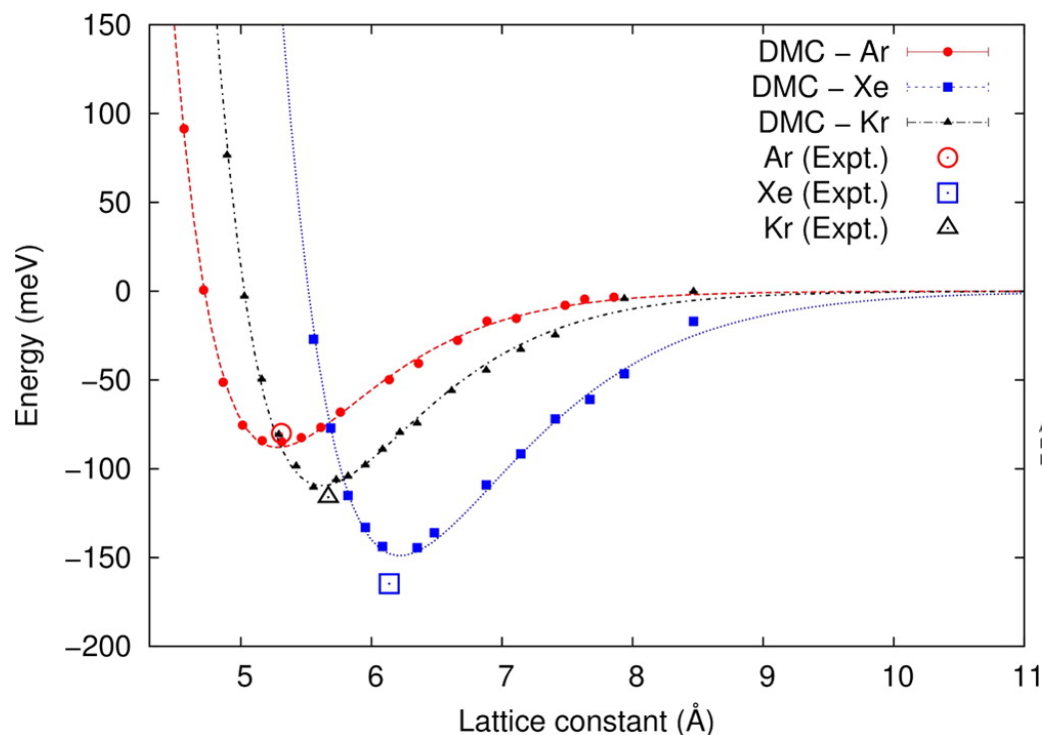
“Quantum Monte Carlo applied to solids” L. Shulenburger & T. R. Mattsson PRB **88** 245117 (2013) <http://dx.doi.org/10.1103/PhysRevB.88.245117> Editors’ Suggestion.



# van der Waals interactions

Extensive DMC calculations for noble gas solids, clusters and DNA+intercalating drug molecule. Drug molecule requires 3 body corrections in vdw DFT to  $\sim$ match DMC results, while some other correction schemes overbind. (PBE DFT does not bind)

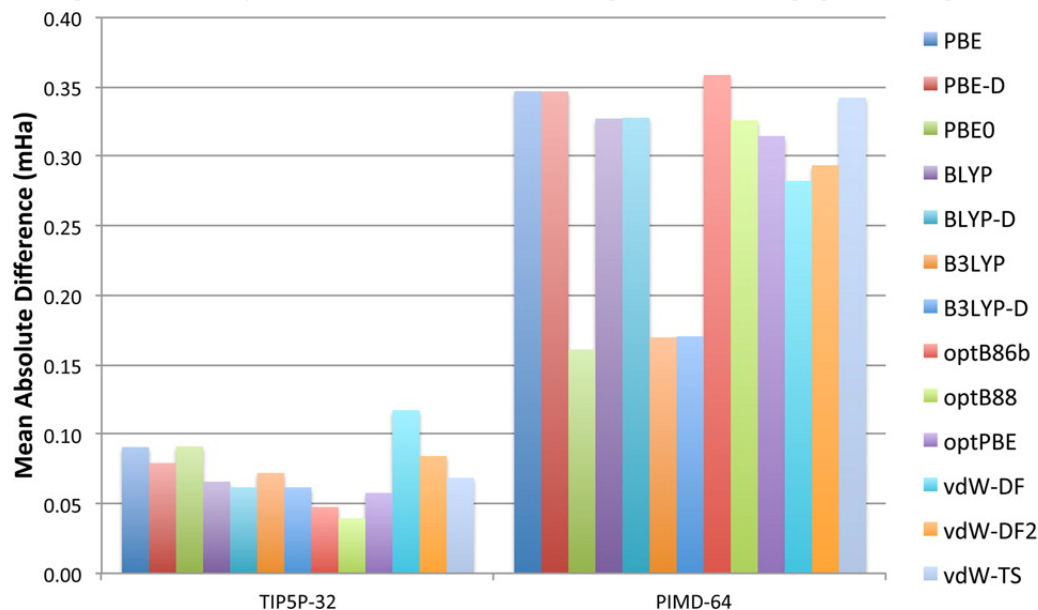
A. Benali, L. Shulenberger, N. A. Romero, J. Kim, O. Anatole von Lilienfeld JCTC (June 2014), <http://dx.doi.org/10.1021/ct5003225>



# Benchmarking density functionals for bulk water

M. A. Morales, J. R. Gergely, J. McMinis, J. M. McMahon, J. Kim, and D. M. Ceperley  
JCTC 10 2355 (2014) <http://dx.doi.org/10.1021/ct500129p>

Analysis of energies of water configurations taken from TIP5P and PIMD (path integral) simulations computed using various DFT functionals vs DMC. Identifies a possible route to optimizing functionals for specific applied problems.

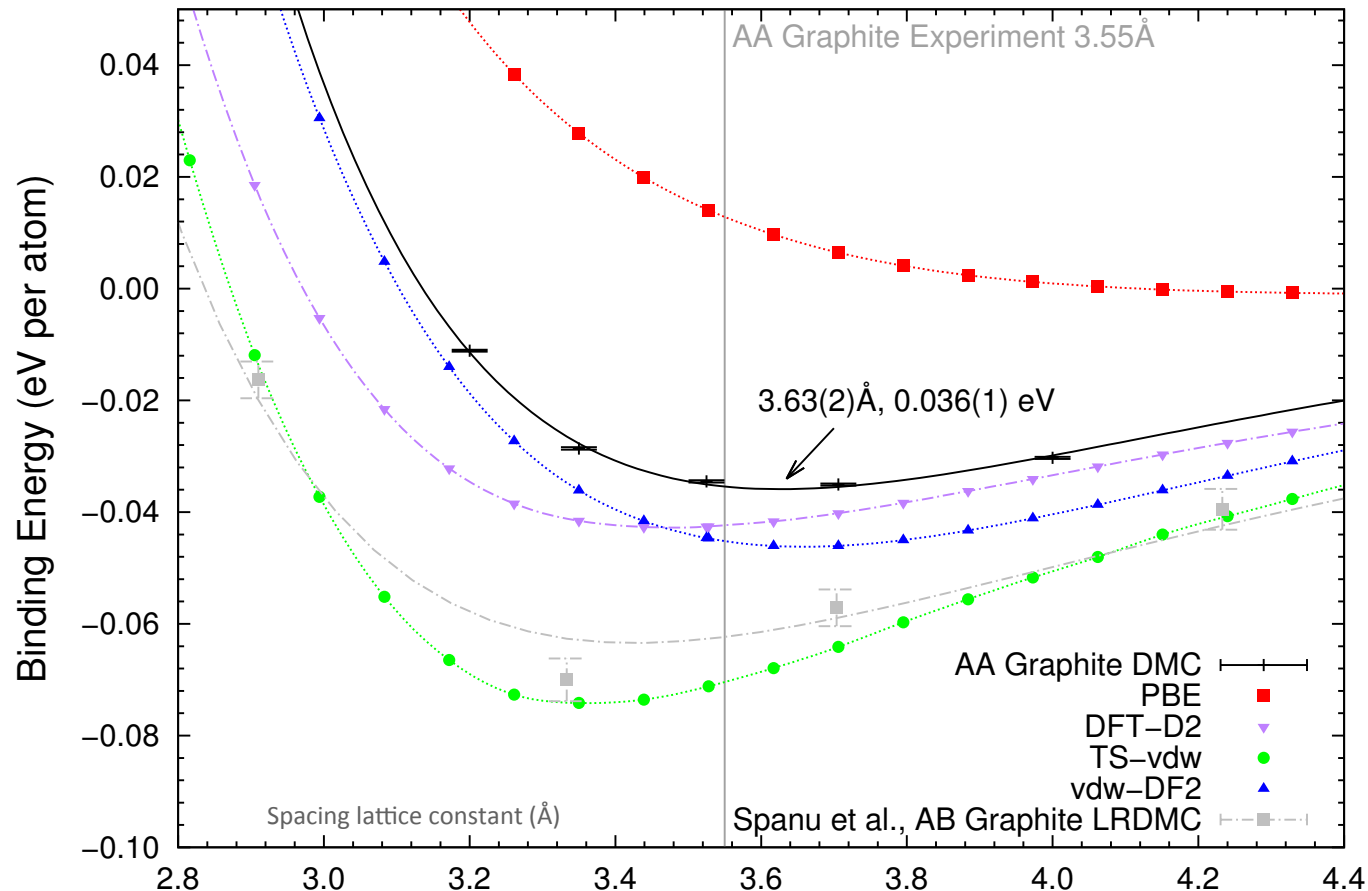


“We found that, while optB88 seems to provide the best description of dispersion in the liquid, the vdW-DF and vdW-DF2 functionals offer the best agreement of all nonhybrid functionals when fully flexible, realistic water configurations are considered.”

# Binding and diffusion of lithium in graphite

DMC calculations accurately predict the lattice constant and binding energy of A-A graphite relative to A-B graphite. When dilute Li is added, self-consistent van der Waals DFTs outperform empirical schemes due to the importance of charge transfer.

P. Ganesh, J. Kim, C. Park, M. Yoon, F. A. Reboredo, and P. R. C. Kent (submitted).

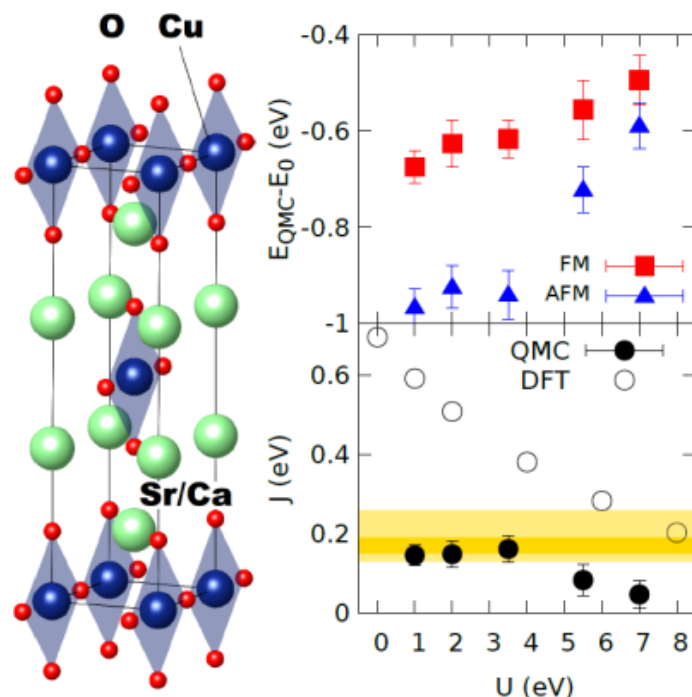


# Computing the exchange constant in cuprates

Within a variational scheme (a non-empirical scheme), DMC predicts exchange constants in good agreement with experiment for  $\text{Ca}_2\text{CuO}_3$ . Indicates promise for describing ground state properties of strongly correlated materials.

K. Foyevtsova, J. T. Krogel, J. Kim, P. R. C. Kent, E. Dagotto, F. A. Reboredo.

Accepted in PRX (2014) and <http://arxiv.org/abs/1402.5561>

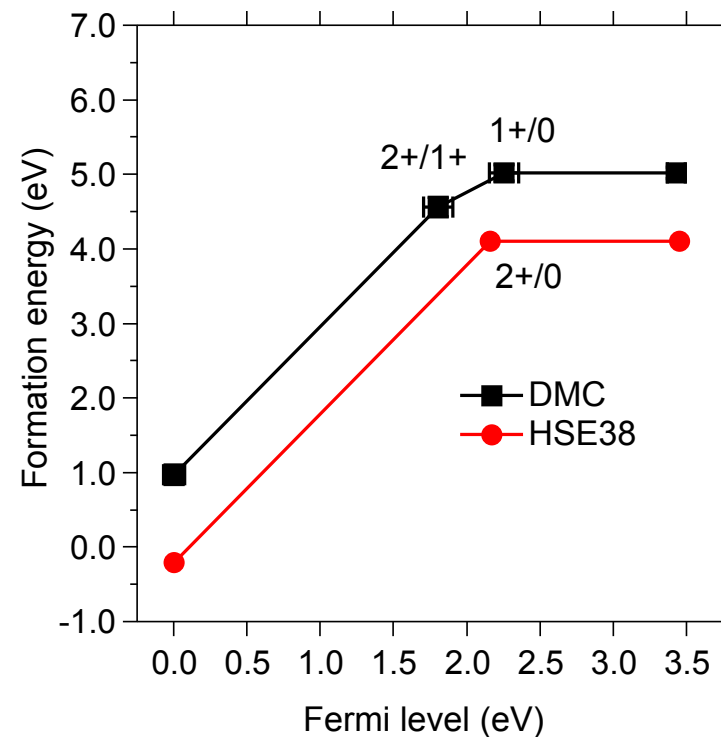


Also see QMC results for  $\text{La}_2\text{CuO}_4$ , with more properties considered including phonons, L. Wagner & P. Abbamonte <http://arxiv.org/abs/1402.4680>

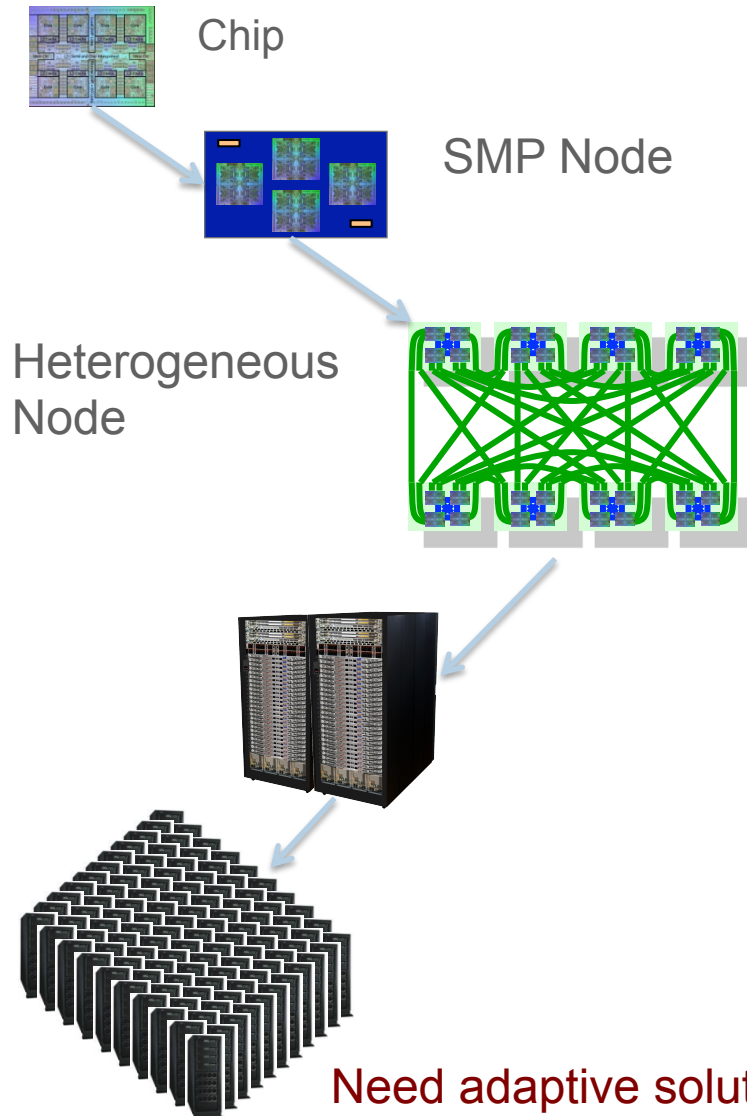
# The vacancy in ZnO

DMC calculations rule out the oxygen vacancy as the source of persistent n-type conductivity in ZnO. Confirms previous DFT predictions but finds (i) much higher oxygen vacancy formation energy than HSE or other DFT approximations, (ii) finds a positive U behavior in contrast to the DFT.

“Ab initio many-body calculations of the oxygen vacancy in ZnO”, J. A. Santana, J. T. Krogel, J. Kim, P. R. C. Kent, and F. A. Reboredo, <http://arxiv.org/abs/1406.3169>



# Challenges



- Balance computational efficiency vs complexity for target accuracy
- Balance memory use vs computational intensity
- Optimize memory access
- Optimize computational kernels and data structures: problem and hardware dependent
- Expose parallelisms: tasks, SIMD
- Optimize communications on nodes and between them
- Handle faults
- Automate all of these and more
- Design adaptive software

Need adaptive solutions for increasingly complex, hierarchical & heterogeneous architectures

# Acknowledgment

- |                     |                       |
|---------------------|-----------------------|
| - Jeongnim Kim.     | Intel (Formerly ORNL) |
| - David Ceperley    | UIUC                  |
| - Paul Kent         | ORNL                  |
| - Luke Shulenburger | SNL                   |
| - Miguel Morales    | LLNL                  |
| - Jaron Krogel      | ORNL                  |
| - Nichols Romero    | ANL                   |

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research was also partly funded by the Early Science Program at Argonne Leadership Computing Facility.

Supported through Predictive Theory and Modeling for Materials and Chemical Science program by the Office of Basic Energy Science (BES), Department of Energy (DOE)."





# Questions



